	INSTITUTO FEDERAL DO NORTE DE MINAS GERAIS Campus Montes Claros Bacharelado em Ciência da Computação

Disciplina: Computação Gráfica	Atividade: TP03 - Space Invaders	Professor: Wagner Ferreira de Barros	Valor: 20pts	Nota:
Data: 05/11/2025	Alunos:			

INSTRUÇÕES

- Este trabalho poderá ser realizado em DUPLAS.
- Todo e qualquer material didático online ou pessoal poderá ser utilizado para a realização desta atividade (e é fortemente aconselhado que seja utilizado). Entretanto, todo material bibliográfico consultado deverá ser citado.
- Cópias de trabalhos feitos por seus colegas, ou trabalhos feitos com uso completo de IA's não serão admitidos e, se identificados, haverá punição com anulação TOTAL ou parcial da nota dos alunos envolvidos.

1 Introdução

Neste trabalho prático, você irá implementar uma versão simplificada do clássico jogo da década de 70, Space Invaders [1], utilizing OpenGL (legacy) e GLUT. O foco será a aplicação de boas práticas de Programação Orientada a Objetos (POO) para gerenciar as entidades do jogo (jogador, alienígenas, projéteis) e o estado geral da aplicação (pontuação, vidas, etc.). Como se trata de um trabalho prático mais completo em OpenGL, você receberá um código base que contém a estrutura inicial do jogo, com a janela, o sistema de coordenadas 2D e o game loop (baseado em glutTimerFunc) já configurados. Seu objetivo será completar a implementação adicionando as entidades do jogo, a lógica de movimentação, a detecção de colisões e o gerenciamento de estado. A Figura 1 ilustra uma tela inspirada no jogo original. Vocês poderão criar com gráficos ainda mais simples.

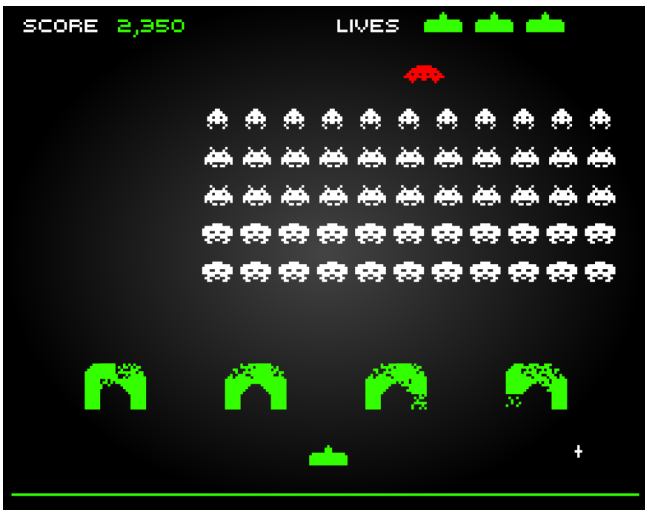

Figura 1: Captura de tela do jogo Space Invaders obtida de .

[1] História sobre o jogo: https://en.wikipedia.org/wiki/Space_Invaders.

2 Objetivos

Os principais objetivos deste trabalho prático são:

- Compreender o funcionamento de uma aplicação gráfica interativa (jogo) usando OpenGL e GLUT.
- Implementar a movimentação de objetos (Jogador, Aliens, Projéteis) e detecção de colisões (AABB - Axis-Aligned Bounding Box).
- Gerenciar o estado do jogo, incluindo o controle do teclado para o jogador e a lógica de movimentação do "enxame"de alienígenas.
- Adicionar um sistema de placar (HUD), controle de vidas e condições de vitória/derrota.
- Estruturar o código-fonte utilizando boas práticas de Programação Orientada a Objetos (POO), criando classes para as diferentes entidades do jogo.

3 Regras do Jogo (Simplificado)

O jogo segue as seguintes regras:

- O jogador controla um "canhão"(um retângulo ou triângulo) na parte inferior da tela.
- O canhão pode se mover horizontalmente (esquerda e direita), controlado pelas teclas de seta (<- e ->).
- O jogador pode disparar projéteis verticalmente para cima (tecla 'espaço').
- Restrição de Tiro: O jogador só pode ter um projéitil ativo na tela por vez.
- Um "enxame"(grid) de alienígenas (ex: 5 linhas x 10 colunas) começa na parte superior da tela.
- O enxame inteiro se move como um bloco, horizontalmente.
- Quando o alienígena mais próximo da borda (esquerda ou direita) atinge o limite da tela, o enxame inteiro desce uma pequena quantidade e inverte sua direção horizontal.
- Em intervalos de tempo regulares (ex: a cada 1 ou 2 segundos), um alienígena aleatório da fileira inferior do enxame dispara um projétil para baixo.
- Colisão (Tiro do Jogador vs. Alien): Se o projétil do jogador atinge um alienígena, o alienígena é removido (morto), o projétil desaparece e o jogador ganha pontos.
- Colisão (Tiro do Alien vs. Jogador): Se o projétil de um alienígena atinge o jogador, o jogador perde uma vida e o projétil desaparece. O jogador reinicia na posição central.
- Condição de Vitória: O jogador vence se destruir todos os alienígenas.
- Condição de Derrota (Game Over): O jogador perde se perder todas as suas vidas (ex: 3) OU se qualquer alienígena do enxame conseguir descer até a altura (posição Y) do jogador.
- Simplificações: Não é necessário implementar os bunkers (barreiras de proteção). A renderização deve usar apenas primitivas do OpenGL (GL_QUADS, GL_TRIANGLES, etc.), sem texturas.

4 Implementação

Você deve completar a implementação do jogo com base no código fornecido. As principais tarefas incluem:

- Estruturar o código com classes (POO), conforme sugestão na Seção 7.
- Implementar a classe Player, com movimentação contínua baseada no estado das teclas e a lógica de disparo (limitado a um tiro).
- Implementar a lógica do AlienSwarm, controlando o movimento do bloco de aliens (horizontal, descida e inversão de direção).
- Implementar a lógica de disparo dos alienígenas (aleatória e em intervalos).
- Implementar a detecção de colisão (AABB) entre projéteis e entidades (aliens/jogador).
- Desenhar o HUD (Placar e Vidas) usando glutBitmapCharacter.
- Gerenciar o estado do jogo (ex: PLAYING, GAME_OVER, WIN), parando as atualizações e exibindo a mensagem apropriada ao final.

4.1 Sugestão para o Controle do Teclado

Para evitar atrasos na movimentação do canhão, é recomendado o uso de variáveis de estado (bools) para verificar se uma tecla está pressionada. O código base já inclui glutKeyboardFunc, glutKeyboardUpFunc, glutSpecialFunc e glutSpecialUpFunc. Use-os para setar true ou false em variáveis de estado (ex: bool g_keyLeftPressed = false;). A verificação (if (g_keyLeftPressed)) deve ser feita dentro da função update, garantindo que o movimento seja contínuo e fluido enquanto a tecla estiver pressionada.

4.2 Detecção de Colisão (AABB)

AABB (Axis-Aligned Bounding Box) é um método simples e eficiente de detecção de colisão para jogos 2D/3D. ”Axis-Aligned”significa que os ”lados”da caixa delimitadora estão alinhados com os eixos do sistema de coordenadas (X e Y, no nosso caso 2D). Basicamente, tratamos todos os objetos (jogador, aliens, projéteis) como retângulos. Para saber se dois retângulos (A e B) estão colidindo, verificamos se eles NÃO estão separados. Dois retângulos A e B não colidem se qualquer uma destas condições for verdadeira:

- A está totalmente à esquerda de B (A.x_max $<$ B.x_min)
- A está totalmente à direita de B (A.x_min $>$ B.x_max)
- A está totalmente acima de B (A.y_min $>$ B.y_max)
- A está totalmente abaixo de B (A.y_max $<$ B.y_min)

Se nenhuma dessas condições for verdadeira, significa que eles estão colidindo. ☞As coordenadas (min/max) dependem de onde está a “âncora” (x, y) do seu objeto, se é no centro, no canto inferior esquerdo, etc. Ajuste a lógica de acordo com sua implementação.

5 Organização do Projeto e Makefile

Para um projeto em C++, é fundamental manter os arquivos organizados. Recomendamos a seguinte estrutura de diretórios:

Código 1: Organização da estrutura de diretórios para o projeto.

```
/seu_projeto
|-- /include        # Arquivos de header (.h ou .hpp) das suas classes
|   |-- Game.h
|   |-- Player.h
|   |-- Alien.h
|   |-- ...
|-- /src            # Arquivos de implementação (.cpp)
|   |-- main.cpp
|   |-- Game.cpp
|   |-- Player.cpp
|   |-- Alien.cpp
|   |-- ...
|-- /bin            # Onde o executável compilado será colocado
|-- /assets         # Arquivos de som (.wav, .mp3)
|-- Makefile        # Arquivo de build
```

Um Makefile simplificado para compilar seu projeto (no Linux/MSYS2) poderia ser:

Código 2: Exemplo de Makefile (Atualizado para SDL2_mixer).

```
# Compilador
CXX = g++
# Flags de compilacao (Modo Debug, C++11, warnings)
CXXFLAGS = -g -Wall -std=c++11
# Diretorios
IDIR = include
SDIR = src
ODIR = obj
BINDIR = bin

# --- Flags de Linkagem Especificas do Sistema ---
# (Descomente a secao para o seu SO)

# Para LINUX (instalado via apt)
# LDFLAGS = -lGL -lglut -lGLU -lSDL2 -lSDL2_mixer

# Para WINDOWS (usando MSYS2/MinGW)
```

```
LDFLAGS = -lGL -lglut -lGLU -lmingw32 -lSDL2main -lSDL2 -lSDL2_mixer

# --- Restante do Makefile ---

# Encontra todos os arquivos .cpp em src
SOURCES = $(wildcard $(SDIR)/*.cpp)
# Gera nomes dos arquivos objeto .o em obj/
OBJECTS = $(patsubst $(SDIR)/%.cpp,$(ODIR)/%.o,$(SOURCES))

# Nome do executavel
TARGET = $(BINDIR)/space_invaders

# Regra principal
all: $(TARGET)

# Regra para linkar o executavel
$(TARGET): $(OBJECTS)
    @mkdir -p $(BINDIR)
    $(CXX) -o $(TARGET) $^ $(LDFLAGS)
    @echo "Build finalizado: $(TARGET)"

# Regra para compilar arquivos .cpp para .o
$(ODIR)/%.o: $(SDIR)/%.cpp
    @mkdir -p $(ODIR)
    $(CXX) $(CXXFLAGS) -I$(IDIR) -c $< -o $@

# Regra para limpar os arquivos compilados
clean:
    rm -f $(ODIR)/*.o $(TARGET)
```

6 Opcional: Adicionando Som com SDL2_mixer

Adicionar efeitos sonoros (tiro, explosão) e música de fundo pode enriquecer o jogo. Uma biblioteca simples para isso é a SDL2_mixer.

6.1 Instalação

- Ubuntu/Debian: sudo apt-get install libsdl2-mixer-dev
- Windows (MSYS2): pacman -S mingw-w64-x86_64-SDL2_mixer

Ao compilar, você precisará adicionar as flags de linkagem corretas, conforme mostrado no Makefile de exemplo (Seção 5).

6.2 Uso Básico

O uso envolve inicializar o áudio, carregar arquivos de som (músicas como Mix_Music e sons curtos como Mix_Chunk) e tocá-los. Um exemplo de código para inicialização e uso pode ser encontrado no Anexo B, na Página 13.

7 Arquitetura Sugerida (POO)

A Figura 2 apresenta uma sugestão de diagrama de classes para organizar o projeto. Recomenda-se fortemente a criação de uma classe base GameObject (seja ela abstrata ou concreta) para unificar as entidades que precisam ser desenhadas e atualizadas, como Player, Alien e Projectile.

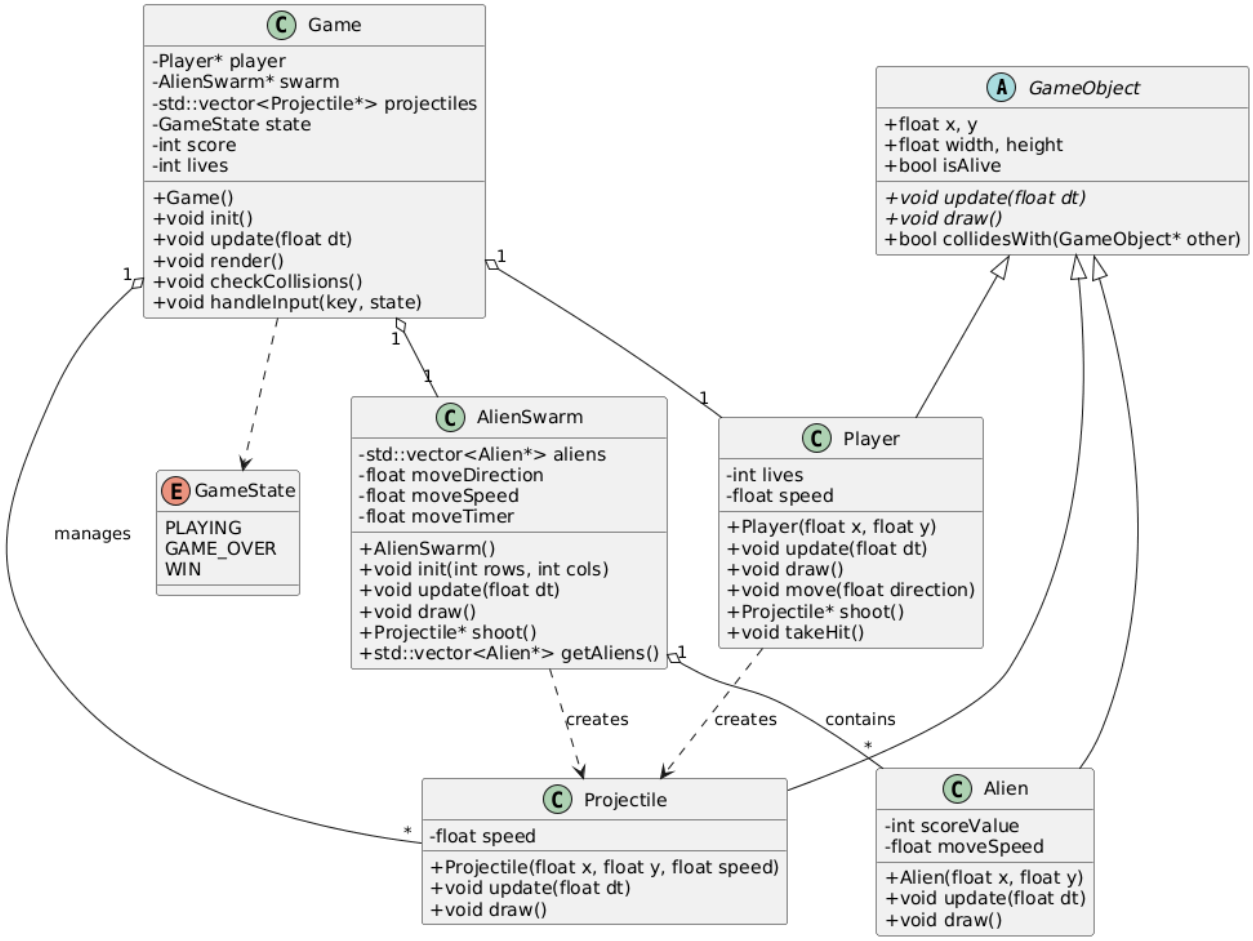

Figura 2: Sugestão de Diagrama de Classes para o Jogo.

8 Critérios de Avaliação

O trabalho será avaliado de acordo com os seguintes critérios:

- Implementação correta da movimentação do jogador e do enxame de aliens.
- Implementação correta da lógica de disparos (jogador e aliens) e detecção de colisões (AABB).
- Gerenciamento de estado funcional (vidas, pontuação, telas de vitória/derrota).
- Qualidade do código, incluindo clareza, organização e, principalmente, o uso apropriado de Programação Orientada a Objetos.
- Documentação do código, incluindo comentários explicativos sobre o funcionamento das classes e métodos principais.

9 Código Base

O código base (inicial) para implementação do TP03, que deve ser refatorado para a arquitetura POO, encontra-se no Anexo A, na Página 8.

10 Conclusão

Este trabalho prático proporcionará uma experiência rica na implementação de um jogo gráfico, abordando desde a movimentação de objetos até a gestão de estados e eventos, com um foco importante na organização do código usando Programação Orientada a Objetos. Boa sorte na implementação!

A Anexo A: Código Base (Ponto de Partida)

Listagem 3: Código Base (Ponto de Partida) do Jogo Space Invaders.

```
// Código base (inicial) para implementação do TP03 de CG
// (Space Invaders)
#include <GL/glut.h>
#include <stdio.h> // Para printf

// Tamanho da janela
const int width = 800;
const int height = 600;

// ----- TODO: Mover estado para classes (POO) -----
// Globais provisorias para o estado do jogo.
// O objetivo e substituir isso por classes!

// Estado das teclas
bool keyState[256];
bool specialKeyState[256];
// Exemplo: Posicao do Jogador
float playerX = width / 2;
float playerY = 50;
float playerSize = 30;
// Exemplo: Tiro do Jogador
bool playerShotActive = false;
float playerShotX = 0;
float playerShotY = 0;
// ------------------------------------------------
// Funcao para desenhar o HUD (Vidas e Pontos)
void drawHUD() {
    // ----- TODO: Implementar -----
    // Exemplo de como desenhar texto:
    // (Lembre-se de salvar/restaurar a matriz de projecao)
    // glColor3f(1.0f, 1.0f, 1.0f);
    // glRasterPos2f(10, height - 20); // Posicao
    // const char* text = "Score: 0";
    // for (const char* c = text; *c != '\0'; c++) {
    //     glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, *c);
    // }
}
```

```
38
39 void display() {
40     glClear(GL_COLOR_BUFFER_BIT);
41     glMatrixMode(GL_MODELVIEW);
42     glLoadIdentity();
43     // ----- TODO: Chamar metodos de desenho (draw) das classes -----
44
45     // Exemplo de desenho do jogador (substituir por player.draw())
46     glPushMatrix();
47     glTranslatef(playerX, playerY, 0);
48     glColor3f(0.0f, 1.0f, 0.0f); // Verde
49     glBegin(GL_QUADS);
50         glVertex2f(-playerSize / 2, -playerSize / 2);
51         glVertex2f( playerSize / 2, -playerSize / 2);
52         glVertex2f( playerSize / 2,  playerSize / 2);
53         glVertex2f(-playerSize / 2,  playerSize / 2);
54     glEnd();
55     glPopMatrix();
56
57     // Exemplo de desenho do tiro (substituir por shot.draw())
58     if (playerShotActive) {
59         glPushMatrix();
60         glTranslatef(playerShotX, playerShotY, 0);
61         glColor3f(1.0f, 1.0f, 1.0f); // Branco
62         glBegin(GL_QUADS);
63             glVertex2f(-3, -8);
64             glVertex2f( 3, -8);
65             glVertex2f( 3,  8);
66             glVertex2f(-3,  8);
67         glEnd();
68         glPopMatrix();
69     }
70
71     // ----- TODO: Desenhar Aliens (swarm.draw()) -----
72
73     // Desenha o HUD por ultimo
74     drawHUD();
75     glutSwapBuffers();
76 }
77
```

```
78  void update(int value) {
79      // ----- TODO: Chamar metodos de atualizacao (update) das classes -----
80
81      // Logica de input (mover para a classe Player)
82      float playerSpeed = 8.0f;
83      if (specialKeyState[GLUT_KEY_LEFT] && playerX > playerSize / 2) {
84          playerX -= playerSpeed;
85      }
86      if (specialKeyState[GLUT_KEY_RIGHT] && playerX < width - playerSize / 2) {
87          playerX += playerSpeed;
88      }
89
90      // Logica do tiro (mover para a classe Player/Game)
91      if (keyState[' '] && !playerShotActive) {
92          playerShotActive = true;
93          playerShotX = playerX;
94          playerShotY = playerY + 20;
95          // playSound(gSomDisparo); // (Opcional, se usar SDL_mixer)
96      }
97
98      // Atualiza posicao do tiro (mover para classe Projectile)
99      if (playerShotActive) {
100         playerShotY += 15.0f;
101         // Desativa o tiro se sair da tela
102         if (playerShotY > height) {
103             playerShotActive = false;
104         }
105     }
106
107     // ----- TODO: Chamar swarm.update() -----
108     // ----- TODO: Chamar game.checkCollisions() -----
109     // ----- TODO: Chamar game.checkGameState() -----
110
111
112     // Re-agendar a proxima atualizacao
113     glutTimerFunc(16, update, 0); // ~60 FPS
114
115     // Forcar redesenho
116     glutPostRedisplay();
117 }
```

```
118 // Callbacks de Teclado (Estado)
119 void handleKeysDown(unsigned char key, int x, int y) {
120     keyState[key] = true;
121 }
122
123 void handleKeysUp(unsigned char key, int x, int y) {
124     keyState[key] = false;
125 }
126
127 void handleSpecialKeysDown(int key, int x, int y) {
128     specialKeyState[key] = true;
129 }
130
131 void handleSpecialKeysUp(int key, int x, int y) {
132     specialKeyState[key] = false;
133 }
134
135 // Funcao para configurar a projecao 2D
136 void reshape(int w, int h) {
137     // Previne divisao por zero
138     if (h == 0) h = 1;
139
140     // Usa a janela inteira para renderizar (default glut)
141     glViewport(0, 0, w, h);
142
143     // Configura a matriz de projecao
144     glMatrixMode(GL_PROJECTION);
145     glLoadIdentity();
146     // Define o sistema de coordenadas 2D
147     // Origem (0,0) no canto inferior esquerdo
148     gluOrtho2D(0.0, width, 0.0, height);
149     // DICA: ajuste de acordo com a razão de aspecto da janela
150
151     // Volta para a matriz ModelView
152     glMatrixMode(GL_MODELVIEW);
153     glLoadIdentity();
154 }
155
156
157
```

```
158 void Init(int argc, char** argv)
159 {
160     glutInit(&argc, argv);
161     // Usa Double Buffer (para animacao) e RGB
162     glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
163     glutInitWindowSize(width, height);
164     glutInitWindowPosition(100, 100); // Posicao
165     glutCreateWindow("TP03 - Space Invaders");
166     // Cor de fundo (preto)
167     glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
168     // Inicializa estados das teclas
169     for(int i=0; i<256; i++) {
170         keyState[i] = false;
171         specialKeyState[i] = false;
172     }
173 }
174
175 int main(int argc, char** argv) {
176     Init(argc, argv);
177     // Registra os callbacks
178     glutDisplayFunc(display);
179     glutReshapeFunc(reshape);
180     glutTimerFunc(16, update, 0); // Inicia o game loop
181
182     // Callbacks de teclado (baseado em estado)
183     glutKeyboardFunc(handleKeysDown);
184     glutKeyboardUpFunc(handleKeysUp);
185     glutSpecialFunc(handleSpecialKeysDown);
186     glutSpecialUpFunc(handleSpecialKeysUp);
187
188     // glutIgnoreKeyRepeat(1); // Opcional: ignora repeticao
189
190     // ----- TODO: Chamar game.init() -----
191     // initAudio(); // (Opcional, se usar som)
192
193     glutMainLoop();
194
195     // cleanupAudio(); // (Opcional, se usar som)
196     return 0;
197 }
```

B Anexo B: Exemplo de Código Opcional (SDL2_mixer)

Listagem 4: Exemplo de uso simplificado da SDL2_mixer

```c
#include <SDL2/SDL.h>
#include <SDL2/SDL_mixer.h>
// Globais para os sons
Mix_Music *gMusicaFundo = NULL;
Mix_Chunk *gSomDisparo = NULL;
Mix_Chunk *gSomExplosao = NULL;

bool initAudio() {
    // Inicializa o SDL Audio
    if (SDL_Init(SDL_INIT_AUDIO) != 0) {
        printf("Falha ao inicializar SDL: %s\n", SDL_GetError());
        return false;
    }
    // Inicializa o SDL_mixer
    // (Frequencia, formato, canais, tamanho do chunk)
    if (Mix_OpenAudio(44100, MIX_DEFAULT_FORMAT, 2, 2048) != 0) {
        printf("Falha ao inicializar SDL_mixer: %s\n", Mix_GetError());
        SDL_Quit();
        return false;
    }

    // Carrega os arquivos de som
    // Mix_LoadMUS para musicas longas (ex: .mp3, .ogg)
    gMusicaFundo = Mix_LoadMUS("assets/musica.mp3");
    // Mix_LoadWAV para sons curtos (ex: .wav)
    gSomDisparo = Mix_LoadWAV("assets/shot.wav");
    gSomExplosao = Mix_LoadWAV("assets/explosion.wav");
    if(gMusicaFundo == NULL || gSomDisparo == NULL || gSomExplosao == NULL) {
        printf("Falha ao carregar midia: %s\n", Mix_GetError());
        // Nao e critico, o jogo pode rodar sem som
    }
    // Toca a musica de fundo em loop (-1 = loop infinito)
    if(gMusicaFundo) {
        Mix_PlayMusic(gMusicaFundo, -1);
    }
    return true;
}

```

```
39 // Funcao para tocar um som curto
40 void playSound(Mix_Chunk *som) {
41     if (som != NULL) {
42         // Toca o som em um canal disponivel
43         // (canal = -1 para o primeiro livre, 0 = nao repetir)
44         Mix_PlayChannel(-1, som, 0);
45     }
46 }
47
48 void cleanupAudio() {
49     // Libera a memoria dos sons
50     Mix_FreeMusic(gMusicaFundo);
51     Mix_FreeChunk(gSomDisparo);
52     Mix_FreeChunk(gSomExplosao);
53     gMusicaFundo = NULL;
54     gSomDisparo = NULL;
55     gSomExplosao = NULL;
56
57     // Fecha o mixer e o SDL
58     Mix_CloseAudio();
59     SDL_Quit();
60 }
61
62 // No seu main:
63 // main() {
64 //     ...
65 //     Init(argc, argv);
66 //     initAudio(); // Chame apos o Init do GLUT
67 //     ...
68 //     // Quando o jogador atirar:
69 //     // playSound(gSomDisparo);
70 //     //     // Quando um alien explodir:
71 //     // playSound(gSomExplosao);
72 //     ...
73 //     glutMainLoop();
74 //     cleanupAudio(); // Chame apos o glutMainLoop
75 //     return 0;
76 // }
```